VIRTUAL DISTRIBUTED SECURITY SYSTEM

[01] This application relates to and claims priority from U.S. Provisional Application serial number 60/329,796, filed October 16, 2001, and U.S. Provisional Application serial number 60/___,___ (Attorney Docket Number MS188903.2), filed October 19, 2001, each of which is herein incorporated by reference.

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

[02] The present invention relates to the field of computer security systems. More particularly, the invention provides methods and devices for providing security to distributed computer systems.

DESCRIPTION OF RELATED ART

[03] Security frameworks have been developed to protect data transmitted in distributed computing systems. Existing security frameworks have an assortment of degrees of privacy, security, adaptability and scalability. The Kerberos system, for example, provides secure communications by users sharing a key with a third party. In order to conduct secure communications, each party connect to the third party and utilize the key issued by the third party. Among other disadvantages, the Kerberos system allows the third party to track the identities of users who are communicating with each. Furthermore, the third party has the ability to decrypt messages because the third party issues the keys. The Kerberos security model is fixed; that is, administrators have limited flexibility in deployment options.

[04] Other existing security systems have merits and limitations. For example, security systems that utilize public key and private key infrastructures can include time-consuming and expensive encryption and decryption steps. Time consuming encryption and decryption steps can limit the scalability of a security system if they

are performed too frequently. As well, PKI infrastructures often have revocation issues and many don't address the issue of management of multiple trust roots or cross-certification. Most solutions are designed for specific purposes, e.g., SSL or IPsec. Existing security systems have also generally focused on specific cryptographic technologies. For example, Kerberos uses symmetric keys and PKI uses public keys.

[05] There exists a need in the art for a generic security framework, for use with distributed computing systems, that is security protocol independent and that can be scaled for use with wide area networks, such as the Internet, and that is independent of the underlying cryptographic mechanisms being used.

BRIEF SUMMARY OF THE INVENTION

[06] The present invention overcomes one or more problems and limitations of the prior art by providing a generic security framework that is transport and security protocol independent and that can support multiple cryptographic technologies. The security framework utilizes a security policy which may describe the components of the framework, their properties, capabilities, requirements, and interactions. The policy is expressed using a security policy language. The security policy language allows different security components to be defined and configured without changing application code. Using this mechanism, a specific set of security services can be defined which meet the needs of a specific deployment. Furthermore, the security components allow the security services to be partitioned to increase both flexibility and scalability. By capturing the security semantics within a language, the underlying implementation is abstract thereby creating a "virtual distributed security system." Furthermore, with the security semantics expressed in language, the underlying protocols are abstracted allowing the security runtime to support different platforms, technologies, and protocols. Finally, with a security policy definition in language form, it is possible to construct proofs of the security system.

BRIEF DESCRIPTION OF THE DRAWINGS

[07] The present invention is illustrated by way of example and not limited in the accompanying figures in which like reference numerals indicate similar elements and in which:

[08] Figure 1 illustrates an exemplary distributed computing system operating environment;

[09] Figure 2 illustrates an architecture of a virtual distributed security system, in accordance with an embodiment of the invention;

[10] Figure 3 illustrates a sample set of security components that may be defined by a security policy, in accordance with an embodiment of the invention;

[11] Figure 4 illustrates a license directory, in accordance with an embodiment of the invention;

[12] Figure 5 shows a method that may be used by a directory to verify the status of licenses, in accordance with an embodiment of the invention;

[13] Figure 6 illustrates how an application using a virtual distributed security system can communicate with an existing security component, in accordance with an embodiment of the invention;

[14] Figure 7 illustrates elements of a security policy that may be utilized for permission processing, in accordance with an embodiment of the invention;

[15] Figure 8 illustrates a security policy, in accordance with an embodiment of the invention;

[16] Figure 9 illustrates a security policy that specifies how to partition service components, in accordance with an embodiment of the invention;

[17] Figure 10 illustrates a security policy that specifies how partitioned service components maybe deployed to systems, in accordance with an embodiment of the invention; and

[18] Figure 11 illustrates a method of sending secure messages in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[19] Aspects of the present invention are suitable for use in a variety of distributed computing system environments. In distributed computing environments, tasks may be performed by remote computer devices that are linked through communications networks. Embodiments of the present invention may comprise special purpose and/or general purpose computer devices that each may include standard computer hardware such as a central processing unit (CPU) or other processing means for executing computer executable instructions, computer readable media for storing executable instructions, a display or other output means for displaying or outputting information, a keyboard or other input means for inputting information, and so forth. Examples of suitable computer devices include hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCS, minicomputers, mainframe computers, and the like.

[20] The invention will be described in the general context of computer-executable instructions, such as program modules, that are executed by a personal computer or a server. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types. Typically the functionality of the program modules may be combined or distributed as desired in various environments.

[21] Embodiments within the scope of the present invention also include computer readable media having executable instructions. Such computer readable media can be

any available media which can be accessed by a general purpose or special purpose computer. By way of example, and not limitation, such computer readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired executable instructions and which can be accessed by a general purpose or special purpose computer. Combinations of the above should also be included within the scope of computer readable media. Executable instructions comprise, for example, instructions and data which cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions.

[22] Figure 1 illustrates an example of a suitable distributed computing system 100 operating environment in which the invention may be implemented. Distributed computing system 100 is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. System 100 is shown as including a communications network 102. The specific network implementation used can be comprised of, for example, any type of local area network (LAN) and associated LAN topologies and protocols; simple point-to-point networks (such as direct modem-to-modem connection); and wide area network (WAN) implementations, including public Internets and commercial based network services such as Microsoft7 Network. Systems may also include more than one communication network, such as a LAN coupled to the Internet

[23] Computer device 104, computer device 106 and computer device 108 may be coupled to communications network 102 through communication devices. Network interfaces or adapters may be used to connect computer devices 104, 106 and 108 to a LAN. When communications network 102 includes a WAN, modems or other means for establishing a communications over WANs may be utilized. Computer devices 104, 106 and 108 may communicate with one another via communication network 102 in

ways that are well known in the art. The existence of any of various well-known protocols, such as TCP/IP, Ethernet, FTP, HTTP and the like, is presumed.

[24] Computers devices 104, 106 and 108 may exchange content, applications, messages and other objects via communications network 102. In some aspects of the invention, computer device 108 may be implemented with a server computer or server farm. Computer device 108 may also be configured to provide services to computer devices 104 and 106.

[25] Figure 2 illustrates an architecture of a virtual distributed security system in accordance with an embodiment of the invention. Applications 202a, 202b, and 202c use a virtual distributed security system 204 for security-related operations. Virtual distributed security systems 204 may use one or more security policies 210a-210c (for establishing security rules and procedures and implement the security policy with one or more protocols 206a-206c and transports 206a-206c). In one embodiment of the invention, applications 202a, 202b and 202c or other entities may negotiate over the use of a security policy. In particular, the entities may exchange messages to negotiate over portions of security polices 210a-210c to use as a custom security policy. Security policies 210a-210c are described in detail below and be written in a common or different security policy language. In one embodiment, virtual distributed security system 204 is implemented with a set of application programming interfaces (APIs). However, those skilled in the art recognize that there are many alternative mechanisms for implementing virtual distributed security system 204. For example, in other embodiments virtual distributed security system 204 could be fully distributed and/or implemented with a set of services. In yet another embodiment virtual distributed security system 204 could be implemented with a set of Java beans or other Java language derivative.

[26] Security policies 210a-210c may describe the behaviors of components of a system or an overall security semantics of a system. For example, when sending a message, a security policy may require that the message be signed in a specific way or that

mutiple signatures of specific forms must be present, that certain credentials must be presented and that at least a portion of the message is encrypted. In another example, a security policy may identify the steps that must be taken before accessing a service.

[27] The security policy may be authored by a system administrator or an application developers. Instead of a limited number of access control rules like prior art methods, aspects of the present invention allow users and applications to create new access and other rules by creating or modifying a security policy. For example, for a particular service, an adminstrator may want users to have a special read access. In prior art systems, the rights that may be utilized are hard coded within the security system. For example, Windows NT operating systems has 32 defined permission rights. With the present invention, the administrator can define new rights by defining or editing a security policy. The secuirty policy may be capability based, i.e, an application may define a capability and virtual distributed security system 202 may provide that capability.

[28] Security policies 210a-210c may be authored in a security policy language. The security policy language may be used to describe the security components, their properties, capabilities, requirements, interaction semantics and other security aspects. Those skilled in the art will appreciate that such a language could be expressed in many different forms. In some embodiments, the security language may be a logic-based language, such as Prolog or some derivative. In other embodiments, the security language may be rule-based or a procedural based. Of course, the security language may also be any combination of logic-based, rule based or procedural based. In one embodiment, security policies 210a-210c are implemented with extensible markup language (XML) documents.

[29] Figure 3 illustrates a sample set of security components 306a-306i that may be defined by a security policy. Security components 306a-306i may be implemented with library routines, separate services, or other mechanisms. An application 302 is

interacting with a service 304. Security components 306a-306i provide a security environment for this communication.

[30] An identity component 306a may provide a mechanism to authenticate a principal and provide authoritative proof of the identity. Proof of an identity may be established with a credential, ticket, license, or some other mechanism acceptable to identity component 306a. Aspects of the present invention are described below with respect to licenses. A license contains a set of assertions and is signed by an authority. One skilled in the art will appreciate that in alternative embodiments, credentials, tickets or other mechanisms may be utilized.

[31] An admission component 306b may provide a mechanism for regulating access to a trust domain of service 304. Admission component 306b may map external credentials to internal credentials. For example, application 302 may provide public key credentials to admission component 306b, and admission component 306b may return symmetric key credentials to application 302.

[32] A permission component 306c may provide a mechanism for pre-fetching rights, capabilities, or other access control information. Permission component 306c allows service 304 to compare signed credentials received from permission component 306c with the required rights/capabilities for the operation requested by application 302. Comparing rights is generally more efficient for service 304 than "fetching" rights, particularly in embodiments where rights do not change.

[33] A shared key component 306d may provide a mechanism to securely share secrets between service 304, security components 306, and other services that exist within a trust domain of service 304. For example, if the admission service 306b returned a symmetric key license, it may be encoded using a shared key for the trust domain. Service 304 can obtain the shared key used by admission service 306b from shared key service 306d and use the shared key to decode the license.

[34] A revocation component 306e may provide a mechanism for tracking revocation. For example, credentials issued outside the trust domain might be monitored for revocation, for example, using certificate revocation lists (CRLS). Revocation component 306e may also monitor usage within the trust domain and revoke credentials, tickets, licenses, or other mechanisms that are being misused. For example, an administrator can set limits on licenses for application 302 such as call limits, time limits, or frequency limits. A call limit may limit the number of times a license may be utilized. A time limit may limit a license to a specific time period and a frequency limit may limit the number of times a license can be utilized within a given time period. If application 302 exceeds these limits, the license can be revoked. When coupled with a metering system, this provides a mechanism to limit the exposure of a denial of service attack, especially if it is accidental. This mechanism is also important when dealing with server farms. In prior art systems, limits are typically divided by the number of servers and allocated accordingly. For example, if there are 10 servers and there is a call limit of 20, each individual server will have a call limit of 2. In other prior art systems, the call limit is applied to each server. As a result, with 10 servers and a call limit of 20, the actual call limit can be 200. Revocation component 306e may track the activity of all of the servers and ensure that the call limit is not exceeded. Of course, propagation delays between servers and revocation component 306e or a metering system may cause a call limit to be slightly exceeded in some cases.

[35] A trust component 306f may provide a mechanism for managing trust relationships. Trust component 306f may utilize a trust policy or trust statement to indicate how an entity is trusted and/or the extent to which an entity is trusted. A trust relationship allows a first entity to trust a second entity when the first entity trusts an authority that issues a license to the second entity. In some embodiments, entities may define the scope of the trust. For example, a first entity may trust a second entity for transactions with certain URLs. Trust relationships may contain a cryptographic component. For example, with public keys embodiments, certification chains are

processed and, eventually, a certificate may be signed by a trusted authority. This mechanism maintains the trusted identities and their certificates so that they cannot be altered without proper authorization. Similarly, a party may not trust the root of another party's certificate, but may countersign or trust a countersignature on the certificate (cross certification). In many circumstances trusts may be shared across different applications and services or even instances (e.g. farms) of a service. In such an environment trust component 306f provides a mechanism for the trusting parties to be notified or obtain updates including additions, changes, or deletions of trusted parties. Those skilled in the art recognize that there can be multiple mechanisms for distributing updates, including, but not limited to, polling and eventing and a security policy may make selections among the available options.

[36] Trust component 306f may provide a trust mechanism that spans enterprises or organizations. For example, application 302 may work with service 304 in a specific contract to deal with parts ordering and tracking. While application 302 and service 304 may be running at different companies, application 302 and service 304 may both also interact another entity, such as a parts company. Application 302 and service 304 may both choose to trust any entity that the parts company specifies as trusted (e.g. its sub-contractors).

[37] A store component 306g may provide a mechanism for storing, retrieving, encrypting and managing credentials, such as licenses. Store component 306g may be used as a lockbox (personal storage), a directory, a registry, an escrow or other storage mechanism. Furthermore, store component 306g can verify the stored licenses, including periodic revocation checks and countersign the credentials allowing parties that trust store component 306g to optimize processing and skip validation checks.

[38] In one embodiment, the functionality of store component 306g may be implemented with a directory, such as directory 402 shown in figure 4. The owners of licenses may retrieve and use the licenses as they are needed. Directory 402 includes a memory module 404 for storing a plurality of licenses. Memory module 404 is shown with

three licenses 406, 410 and 414 belonging to a consumer A for illustration purposes only. Moreover, while licenses are shown, one skilled in the art will appreciate that other credentials, such as private keys may be utilized. Memory module 404 may contain licenses belonging to any number of entities and issued by any number of authorities. License 406 was issued by an authority 408, license 410 was issued by an authority 412 and license 414 was issued by an authority 416.

[39] Directory 402 may also include a processor 418 and a verification module 420 that can be used to verify the status of one or more licenses stored in memory module 404. Figure 5 illustrates an exemplary method that processor 418 and verification module 420 may use to verify the status of a license. In step 502, directory 402 stores in a memory at least one license issued by an authority to a license owner. License 406, for example, was issued by authority 408 and is stored in memory module 404. Next, in step 504, directory 402 periodically transmits an identification message of the license to the authority. The directory may be configured to transmit identification messages hourly, daily, weekly or at any other interval. In one embodiment of the invention, the interval is determined by the authority that issued the license. For example, an authority may desire to have updates more frequently for more sensitive licenses. The identification messages may include a license identification number, e.g., "156DVX22" for license 406. The authority may use the license identification number to determine whether the license has been revoked, replaced or modified. In an alternative embodiment, one or more authorities periodically transmit status messages to directory 402 without first receiving an identification message. For example, an authority may send a status message to a directory when the status of a license has changed. Polling and synchronization may be used when an entity has been disconnected or offline.

[40] In step 506, the directory receives, from the authority, at least one status message indicating the status of the license. The status message may indicate that the license remains valid, has been revoked, has been modified or some other status. In step 508,

it is determined whether the status message includes instructions for modifying the license. If the status message includes modification instructions, the license is modified in step 510. A license may be modified by expanding or limiting its scope or duration or modifying any assertion or condition. Next, in step 512, it is determined whether the license includes instructions for deleting the license. A license may be deleted when it has been revoked by the issuing authority. If the status message indicates that the license is to be deleted, the license is deleted in step 514.

[41] Directory 402 may receive, from the license owner, a request for the license in step 516. The license owner may request the license when the license owner desires to use the license, for example. When the issuing authority has not revoked the license, directory 402 may transmit the license to the license owner in step 518.

[42] Returning to figure 3, an integrity component 306h may provide mechanisms for signing (digitally) portions of a message and verifying integrity and signatures of received messages in accordance with security policies. A confidentiality component 306i may provide mechanisms for encrypting and decrypting portions of a message. One skilled in the art will appreciate that there are many mechanisms that may be used to implement integrity component 306h and confidentiality component 306i and that in some embodiments a security policy makes a selection from the available options.

[43] One of the advantages of abstracting underlying protocols and transports is that abstraction facilitates interoperability with other systems. Figure 6 illustrates how an application 602 using a virtual distributed security system 604 can communicate with an existing security component 606. Interoperability can occur because virtual distributed security 602 may select the most appropriate protocol 610 and transport 612 based on information included in a security policy 608. In one aspect of the invention, protocol 610 may be a composable protocol. Virtual distributed security system 604 may be built as a set of services with an application programming interface (API) that application 602 may utilize. The API allows application 602 and

system virtual distributed security system 604 to communicate with one another through a variety of different mechanisms such as, but not restricted to RPC, SOAP, or shared memory.

[44] Figure 7 illustrates elements of a security policy 700 that may be utilized for permission processing. Policy 700 may define the security rights 702 associated with each service 704 and the mappings of rights to identities 706. Such mappings could be based on many criteria including, but not limited to, name, authority, credential type, associated group, or associated roles. Policy 700 may also define different messages 704a-704c that a service 704 receives and requirements 708 of each of messages 704a-704c. For example, one or more credential, integrity, confidentiality or right requirements may vary by message. That is, it may be required that a message is signed or encrypted, and it may indicate the supported/required algorithms to use. -Policy 700 illustrates an example in which an XML license is required from an admission component 710. Additionally, a requirement may specify which application protocol elements must be signed and those application protocol elements that are required for a message.

[45] A policy language used to create a security policy may define the functionality of a component using security primitives. The security primitives may be in combination with traditional programming elements. Those skilled in the art will recognize that security primitives can take various forms. In one embodiment, the primitives might be high-level constructs like "create a license" or "manage storage of a license" which are controlled by the use of parameters. For example the policy 802, shown in figure 8, for an admission component 804 might represent "create a license of type X for name N using key Y signed by license L." In another embodiment, the primitives might be very low-level constructs such as "fetch the template for type X; replace the 'name' with N; set the 'date' field to '10/24/2001', set the 'authority' field to Z; set the 'key' field to Y; sign fields 'name', 'date', 'authority', and 'key' with license L

using algorithm Y". Of course, other embodiments of the invention may combine different levels of constructs.

[46] Security policies, such as security policy 802 allow a definition of a distributed security system without the writing of code. Among other advantages, a security system that does not require the writing and rewriting of code allows for custom components to be inserted to augment or replace specific steps. Such components could be provided by, but not limited to, native processor code, Java beans or other Java language derivatives, or Microsoft CLR modules.

[47] In one aspect of the invention, each security module may be described independently and a security policy can then indicate how to combine modules, and, how to deploy modules. Such information may be part of the policy, or part of an independent deployment mechanism. Providing flexibility in describing, combining and deploying modules allows a distributed security system to support partitioning, replication, and farming. For example, in figure 9, policy 902 might specify how to partition service components 904a-904g into separate partitions 906a-906c. Those skilled in the art will appreciate that in addition to security related services, the teachings shown in figure 9 may be applied to non-security related services.

[48] Similarly, a policy, such as policy 1000 shown in figure 10 might further specify which partitions 1002a-1002 are deployed to systems 1004a-1004f in a distributed computing systems. Systems 1004a-1004f may be implemented with applications, computing devices, separate processors of a multiprocessor system or other hardware or software. As is shown in figure 10, a single partition may be present on more than one system of the distributed computing system.

[49] A distributed security system may abstract cryptographic objects and operations to make the system independent of the underlying cryptographic technology. Licenses have been described above and may be a credential that is given to another party to identify oneself. Examples of licenses include X.509 certificates and Kerberos

tickets. A "testament" may be proof that a party uses to prove that they own a license or a key within the license. For example, an X.509 certificate contains a public key and a corresponding testament may be the associated private key. Similarly, a Kerberos ticket includes an encrypted symmetric key and the testament may be the symmetric key itself.

[50] A security policy may utilize elements such as licenses and testaments to create a single programming model for securing messages using operations such as sign, verify, encrypt, and decrypt. To sign a message, the license owner, for example, may encrypt a digest of the message with their testament. The holder of a license may verify the signature. Similarly, to encrypt a message, an entity may use a key in the recipient's license. The recipient may decrypt the message using a testament belonging to the recipient. In embodiments that utilize public keys, a message may be encrypted with a generated symmetric key and the symmetric key may be encrypted with a public key from the license.

[51] Figure 11 illustrates an example in which an entity 1102 sends a message 1104 to another entity 1106. Entity 1102 owns a license 1108 and a testament 1110. Similarly, entity 1106 owns a license 1112 and a testament 1114. Entities 1102 and 1106 wish for message 1104 to be secure. Entity 1102 initiates a sign operation 1116 on message 1104 by computing a digest of message 1104 and then encrypting the digest with its testament 1110. Next, entity 1102 may perform an encrypt operation 1124. Entity 1102 may create a key 1118 and encrypt message 1104 with key 1118. Key 1118 may then be encrypted using license 1112, which belongs to entity 1106. The encryption of key 1118 may involve encrypting key 1118 with a key found in license 1112. The encrypted message, the encrypted key, and the signature are transmitted to entity 1106. In other embodiments of the invention, credentials other than keys may be utilized.

[52] Entity 1106 can perform a decrypt operation 1120 to decrypt key 1118 using its testament 1114. Using key 1118, entity 1106 can decrypt message 1104. Next, entity

1106 may perform a verify operation 1122. Entity 1106 may compute a digest of message 1104 and compare the computed digest with the digest sent by entity 1102 using license 1108.

[53] One of the advantages of the present invention is that applications running for entities 1102 and 1106 may be the same, regardless of whether licenses 1108 and 1112 contain public keys, symmetric keys, digest keys, or other forms of credentials. The specifics of the cryptographic algorithms may be handled by the security system, not the application. Handling cryptographic algorithms with a security systems is particularly desirable when heterogeneous credentials are used. That is, licenses 1108 and 1112 use different cryptographic techniques. In traditional systems, applications must be aware of this and be coded specifically for the presence of diverse cryptographic techniques. The present invention abstracts cryptographic techniques and does not require applications to be aware of the cryptographic technologies being used. As a result, cryptographic techniques can be altered at any time simply by altering a security policy.

[54] The disclosed security model can be used to create richer environments. For example, while describing figure 3, it was indicated that admission component 306b might accept public key credentials and generate symmetric key licenses. The logic in admission component 306b or in any other service may be the same regardless of the type of license used. With such a service, the services can have increased scalability by simply using a symmetric key license. The operation of receiving a license and then issuing another license is defined as a re-issuance operation because component 306b is "re-issuing" the sender's license using its own format. In some cases, component 306b, or a similar component, might return a license with the same key. For example, component 306b may accept an X.509 certificate, but return an XML license with the same public key because component 306c only accepts XML licenses.

[55] Re-issuance may form the basis of a delegation operation. A party A may make a specific delegation to another party B. To do this, A may re-issues B's license specifying the delegations (as well as any conditions) and sign the license as the issuing authority. To use this new license, B may be required to present the license, proof of ownership (e.g. a signature), and A's license (if the recipient doesn't have it). The recipient may then verify that B owns the license and the A issued it, and that the delegations in the license correctly correspond to A's license. The abstraction of cryptographic technologies and, specific license formats, allows applications to use a consistent programming model independent of the cryptographic technologies or license formats used at any specific time.

[56] Another use of licenses and key abstraction is service scalability. Existing systems are typically limited when creating special keys for communications because only the client and the service know the key and state on the service limits scalability. The existing approach has limitations when a service is implemented as part of a server farm. Aspects of the present invention allow the service to re-issue the sender's license providing a session key, but encrypting it with a secret key that is shared across the farm, possibly using the shared key service 306d. Consequently, any server in the farm that receives a message can understand and process the message since it can abstract the session key, and no state is required to be maintained on the farm. Similarly problems arise in existing systems when a client is part of a farm. The technique described above may also be used by the client to provide a license to the service to use on its reply so that any client farm member can service the message.

[57] Security credentials may be passed between components or services using the simple object access protocol SOAP. In one embodiment of the invention, credentials, such as licenses, may be passed with SOAP messages by including them in a SOAP "credentials" header. The header may allow for any type of credential to be passed. Message integrity can be associated with SOAP messages by including digital signatures or other integrity mechanisms in a SOAP "integrity" header. For example,

this header can contain XML Signatures as defined in the W3C standard. Message confidentiality can be achieved by associating an XML tag encryption technology with SOAP messages. One such scheme is the XML Encryption standard as defined by the W3C. Confidentiality of message attachments may be achieved by including a manifest (and encryption meta-data) of the encrypted attachments in a SOAP "confidentiality" header.